Filtering by Associations May 16th, 2011 - by Jeremy Evans

3.23.0

As I mentioned in the **last blog post (/2011/05/02/sequel-3230-released/)**, Sequel 3.23.0 shipped with support for filtering by associations. Basically, it allowed you to use associated objects to filter datasets. For example, you could do:

```
artist = Artist[1]
Album.filter(:artist=>artist)
# SELECT * FROM albums WHERE (artist_id = 1)
```

This didn't just work for many_to_one associations, it works for one_to_many associations:

```
album = Album[1]
# album.artist_id => 3
Artist.filter(:albums=>album)
# SELECT * FROM artists WHERE (id = 3)
```

It also works for many_to_many associations:

```
tag = Tag[1]
Album.filter(:tags=>tag)
# SELECT * FROM albums WHERE (id IN
#  (SELECT album_id FROM albums_tags
#   WHERE (tag_id = 1)))
```

Since all three of these examples could be handled by doing:

```
artist.albums
album.artist
tag.albums
```

this support didn't gain you much unless you wanted to combine multiple associations together:

```
Album.filter(:artist=>artist, :tags=>tag)
# SELECT * FROM albums
# WHERE ((artist_id = 1) AND (id IN
#  (SELECT album_id FROM albums_tags
#   WHERE (tag_id = 1))))
```

Corner Case: NULL

Over the last few days, **I've been fixing some corner cases in this support, and expanding it to include new types of usage (https://github.com/jeremyevans/sequel/compare/5e3ace7dea09f9c094c8...5bc13219e3e30e79b1488f47c79e8213c5b3dab1)**. The main corner case in the support that shipped with 3.23.0 is that NULL values weren't handled correctly. So if you did:

```
Album.filter(:artist=>Artist.new)
# SELECT * FROM albums WHERE (artist_id IS NULL)
```

That's not going to give albums related to that artist, it's going to give you albums not related to any artist. The correct query for this usage is:

```
# SELECT * FROM albums WHERE FALSE
```

Because no albums in the database can be related to an artist not in the database. That is the type of query that Sequel now uses in such a case.

New Feature: Excluding

While filtering by associations worked in 3.23.0, you could not exclude by associations, so attempting to invert a dataset that used an association filter would break. That violates how Sequel in general should behave, so the first new related feature is the ability to exclude by associations:

```
Album.exclude(:artist=>artist)
```

This dataset should match all albums not related to this artist. Exclusion has an interesting twist that filtering does not have, in that the naive approach does not work correctly. For example, this SQL is incorrect:

```
# SELECT * FROM albums WHERE (artist_id != 1)
```

The reason it is incorrect is the same reason as the previously mentioned corner case, NULL. This query will not match albums that have no associated artist. So all association exclusion queries end up looking like:

```
# SELECT * FROM albums WHERE ((artist_id != 1) OR (artist_id IS NULL))
```

Another interesting case in regards to NULL when it comes to excluding records happens with many_to_many associations. For example, the following SQL has a subtle bug:

```
Album.exclude(:tags=>tag)
# SELECT * FROM albums WHERE ((id NOT IN
#  (SELECT album_id FROM albums_tags
#   WHERE (tag_id = 1)) OR (id IS NULL))
```

With intelligent schema constraints, this bug is hidden, but if there is a row in the albums_tags table with tag_id = 1 but album_id = NULL, this query will not work correctly for albums that have no associated tags. The fix is fairly simple:

```
# SELECT * FROM albums WHERE ((id NOT IN
#  (SELECT album_id FROM albums_tags
#   WHERE ((tag_id = 1)
#     AND (album_id IS NOT NULL)))
#  OR (id IS NULL)))
```

This was subtlely broken in the 3.23.0 code, as you could end up with a NULL value instead of a FALSE value for many_to_many association filters.

New Feature: Filtering by Array of Multiple Associated Objects

Sequel makes checking if a column is one of a set of values easily by using a hash with a column key and an array of values in a filter:

```
Album.filter(:artist_id=>[1,2,3])
```

Now that Sequel supports filtering by associations, it seemed logical to allow the same behavior for associations, using pretty much the same query:

```
Album.filter(:artist=>[artist1, artist2, artist3])
# SELECT * FROM albums WHERE (artist_id IN (1, 2, 3))
```

Excluding by an array of associated objects is naturally handled as well:

```
Album.exclude(:artist=>[artist1, artist2, artist3])
# SELECT * FROM albums
# WHERE ((artist_id NOT IN (1, 2, 3)) OR (artist_id IS NULL))
```

Corner cases that had to be handled are things like providing an array like:

```
Album.exclude(:artist=>[artist1, artist2, artist3, Artist.new])
```

Basically, the array has to be scanned for NULL values in keys before being used.

New Feature: Filtering by Association Dataset

Sequel has had generic support for filtering using subselects for years:

```
Album.filter(:artist_id=>DB[:artists].
  filter(:name.like('A%')).select(:id))
# SELECT * FROM albums WHERE (artist_id IN
#   (SELECT id FROM artists
#    WHERE (name LIKE 'A%')))
```

Now you can use association datasets to clean that up a bit:

```
Album.filter(:artist=>Artist.filter(:name.like('A%')))
```

Exclusion works as you might expect, so I won't bore you with another similar SQL query.

Full Support

I've only gone over the basics of this support now. For example, I've only showed associations with single keys. Sequel supports composite key associations, and there is full support for composite key associations in this filter support. Additionally, Sequel ships with a many_through_many plugin that allows you to use an arbitrary number of join tables between two models. That plugin also has full support for filtering by associations, and it supports composite keys too. Here's an example of an exclusion by multiple associated objects filter for a many_through_many association through three join tables using composite keys for each:

```
Artist.many_through_many :tags,
  [[:albums_artists, [:b1, :b2], [:c1, :c2]],
   [:albums, [:d1, :d2], [:e1, :e2]],
   [:albums_tags, [:f1, :f2], [:g1, :g2]]],
  :left_ primary_key=>[:a1, :a2]

Artist.exclude(:tags=>[Tag[1, 2], Tag[3, 4]])
# SELECT * FROM artists
# WHERE (((a1, a2) NOT IN
#  (SELECT albums_artists.b1, albums_artists.b2
#   FROM albums_artists
#   INNER JOIN albums ON ((albums.d1 = albums_artists.c1)
#                     AND (albums.d2 = albums_artists.c2))
#   INNER JOIN albums_tags ON ((albums_tags.f1 = albums.e1)
#                           AND (albums_tags.f2 = albums.e2))
#   WHERE (((albums_tags.g1, albums_tags.g2) IN ((1, 2), (3, 4)))
#     AND (albums_artists.b1 IS NOT NULL)
#     AND (albums_artists.b2 IS NOT NULL))))
# OR (a1 IS NULL) OR (a2 IS NULL))
```

Caveat

While the new filtering by associations support makes some queries much easier, by its nature it can only handle simple association relationships. For example, it's possible to have Sequel associations such as:

```
Artist.one_to_many :x_albums, :key=>:album_id, :class=>:Album do |ds|
  ds.filter((x ? :y : :z)=>0)
end
```

This creates a dataset that depends on the x attribute of the instance, and it doesn't work in an association filter:

```
Artist.filter(:x_albums=>album)
```

There's no way for it to work, since there is no Artist instance to operate on. For similar reasons, such associations don't work with eager loading, but you can add an :eager_loader option to the association to

get custom eager loading. There is currently no way to customize the filtering by associations support, it operates strictly on the foreign key relationships. It is possible to add the ability to customize it in the future, but I'll probably wait for such a request before doing the work.